HP 9836 – Notes and Repairs

Martin Hepperle, June 2022

Recently I acquired a HP 9836A with its monochrome monitor. Nothing special for many, but I wanted to have it for extending my HP 9000/200/300 range towards the HP Series-80 systems. The only missing link is now the 9826 (and maybe a color 9836C).

The machine had been offered on E-Bay for a relatively high (according to my valuation) starting price of 290 EUROs. The photographs showed a CRT with very ugly burn-in traces. One could read the text even when the machine was off. Also the left CAPS LOCK key was missing, which was another negative point. At least the seller was honest and did not hide these flaws.

These were probably the reasons that nobody else wanted this machine. I took the risk because I already had a monochrome monitor in storage for more than 5 years but no 9836. And I hoped to replace the missing key cap with a replica or find a “new” one.

Finally, the machine arrived in two parcels, all wrapped in a few kilometers of sticky tape and air bubble wrap and well cushioned with thick cardboards so that nothing was damaged in transit.

The system proved to be an early machine (Serial # 2143 A 00213: the 213th machine manufactured in week 43 of 1981 in the USA) with 64 KB of RAM on the CPL board and of course no MMU. It came with a BASIC 2.0 ROM board. Additionally, a Datacomm and two 256 KB RAM boards (one HP, one Eventide) were installed – all very authentic for its time.

After a visual inspection of all boards, setting the input voltage switch from 220 to 240 Volts and cleaning and mildly lubricating the two mini-disk drives (one original Tandon, one HP manufactured Tandon drive) I powered the machine up and it booted happily into BASIC.

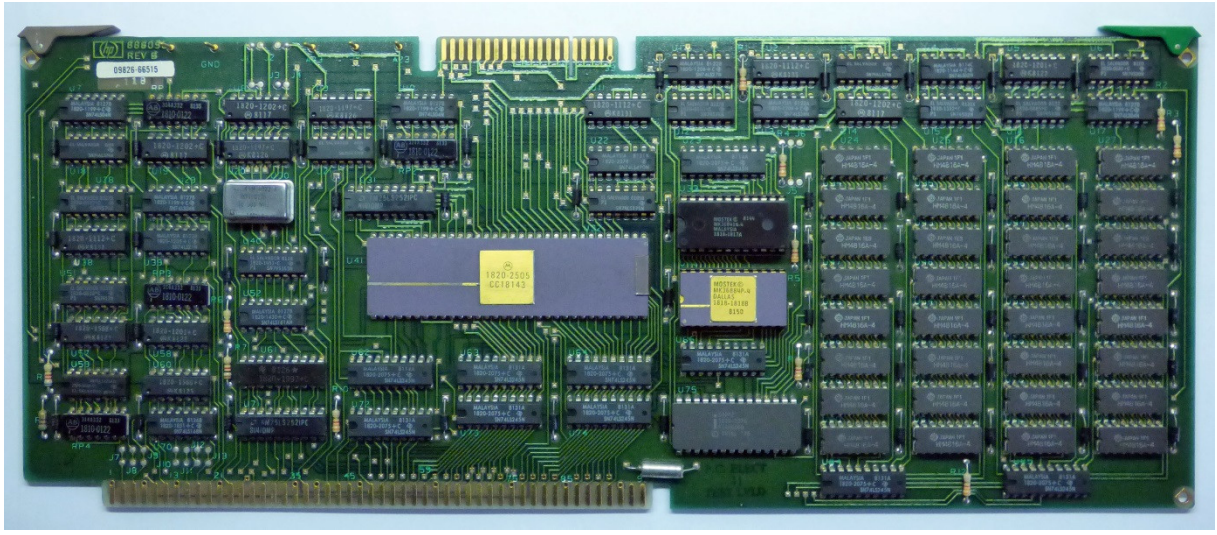

Figure 1: The CPL board 09826-66515.

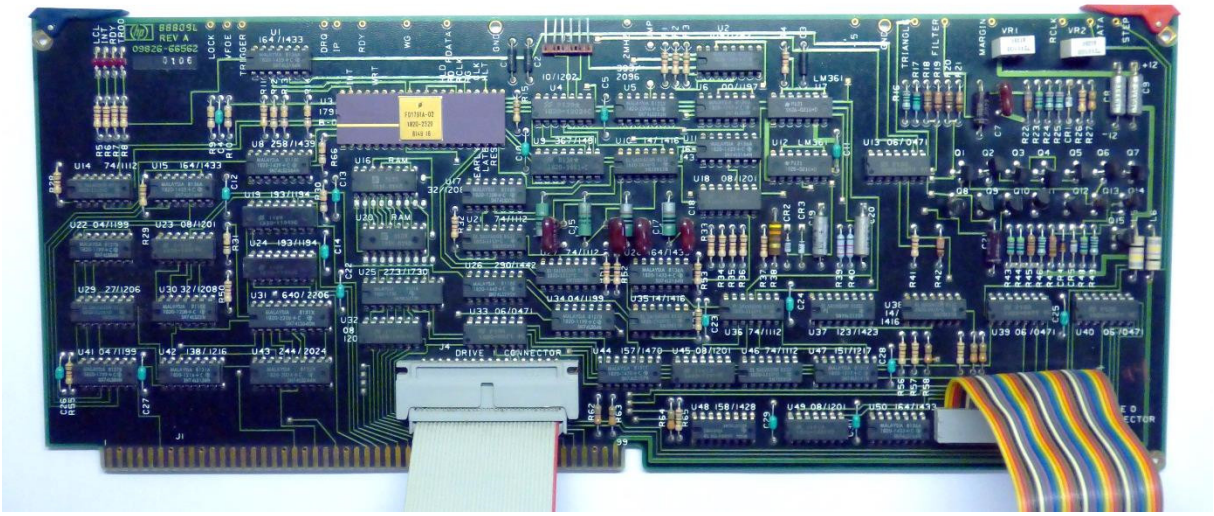

Figure 2: The diskette controller board 09826-66562.

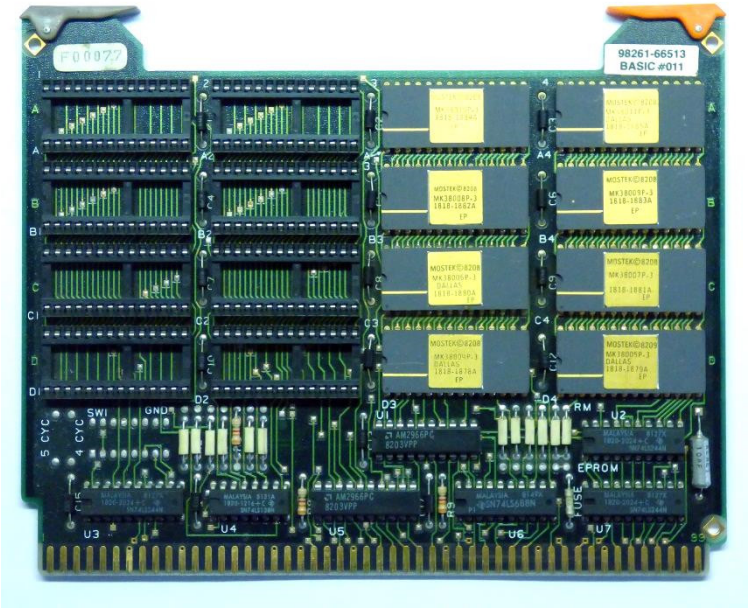

Figure 3: The BASIC 2.0 ROM board 98261-66513.

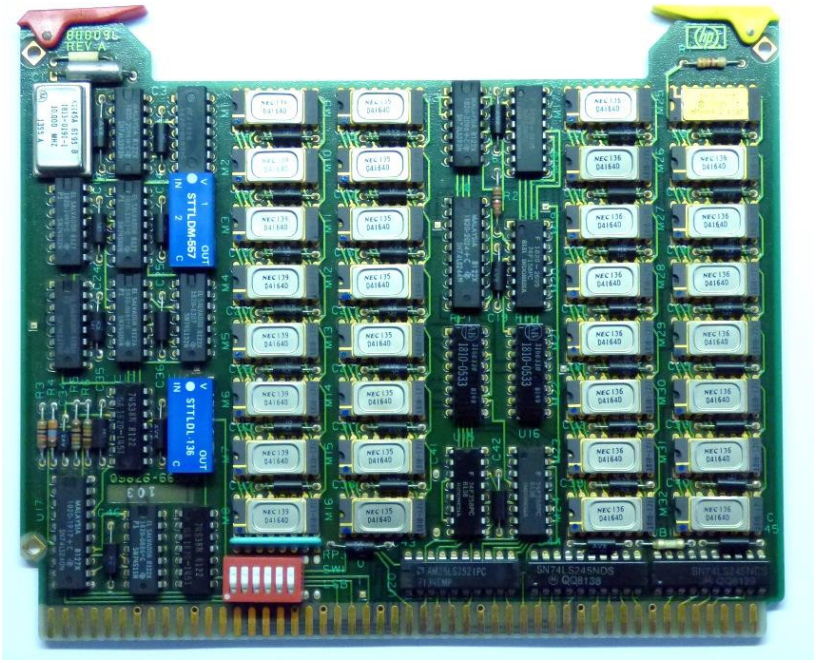

Figure 4: The HP RAM board.

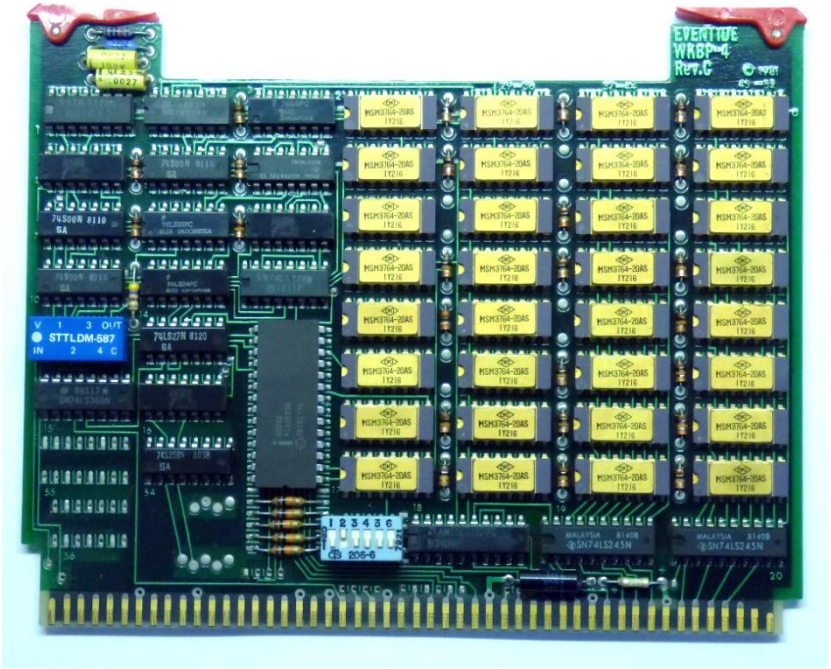

Figure 5: The Eventide RAM board.

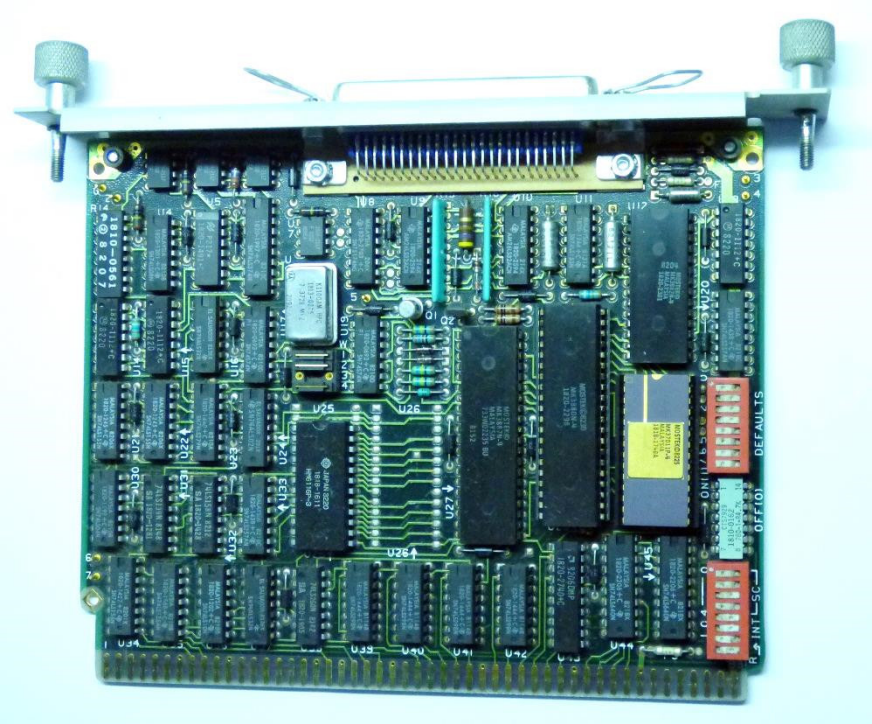

Figure 6: The Datacomm board HP 98622A.

The Knob

The first fault, which I noted, was that the knob was not working. So I removed the keyboard and replaced the burnt out light bulb in in the knob assembly. I had done that before in the Nimitz keyboard of my 9816. All that is needed is a small 6…12V glass light bulb with filament wires and a diameter of about 3 mm. Such bulbs are available for model hobby purposes, e.g. for model railroads.

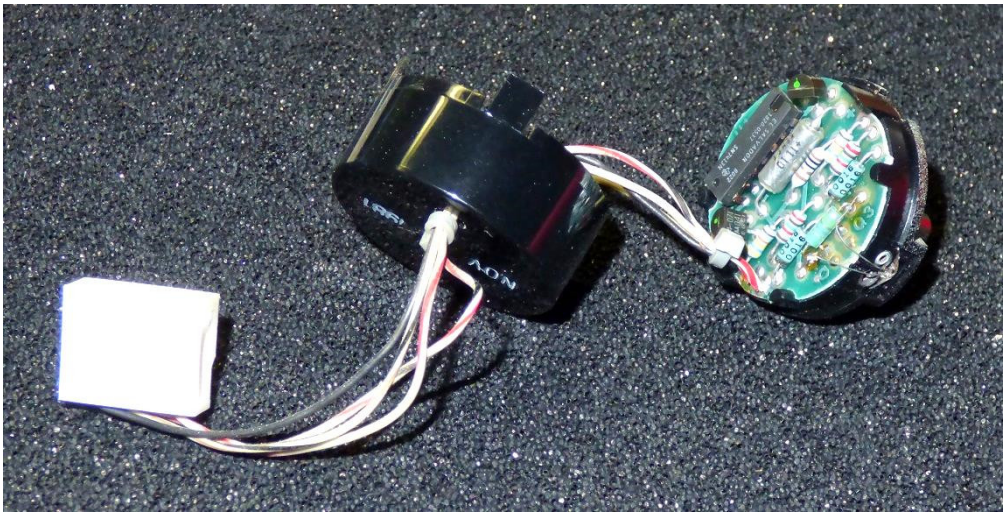

Figure 7: Like on the Nimitz keyboard, the rotary knob is attached to the keyboard PCB by a small edge connector. The black cover can be pulled off after slightly bending the tabs (don't break them, they may be brittle!). The glass bulb is soldered to the circular PCB and a slight press fit into the cavity with a V-shaped sheet metal beam diverter.

The Keyboard

As I had already seen on the photos, the CAPS LOCK key was missing. Indeed, it was not just missing, but the black stem was completely broken off, leaving only the cylindrical shaft of the bare key plunger. To cover the ugly hole, I decided to recreate the key cap.

For mounting the key cap, I drilled two 1 mm diameter holes into the remains of the plunger and carefully glued two short steel wire pins into the plunger. Here I used a steel-filled Epoxy resin glue. This was a slightly tricky operation as I had to avoid damaging the key switch as well as bringing glue into the key mechanism. In preparation of the next steps I also added a very thin layer of Vaseline to the outer sliding part of the plunger.

Figure 8: Keyboard with missing key and steel pins already glued into the plunger.

The key cap could have been created by a CAD redesign and a 3D printer, but I made a silicon rubber mold of the corresponding key cap pulled from a Nimitz keyboard. For this step, the template key cap was suspended upside down on a thin steel strip and the rubber slowly cast into a plastic cup. A larger casting hole and smaller venting holes at the four corners were added for allowing trapped air to escape (it would have been better to add these to the cap before casting the silicone, but I did not want to glue something to the original cap.)

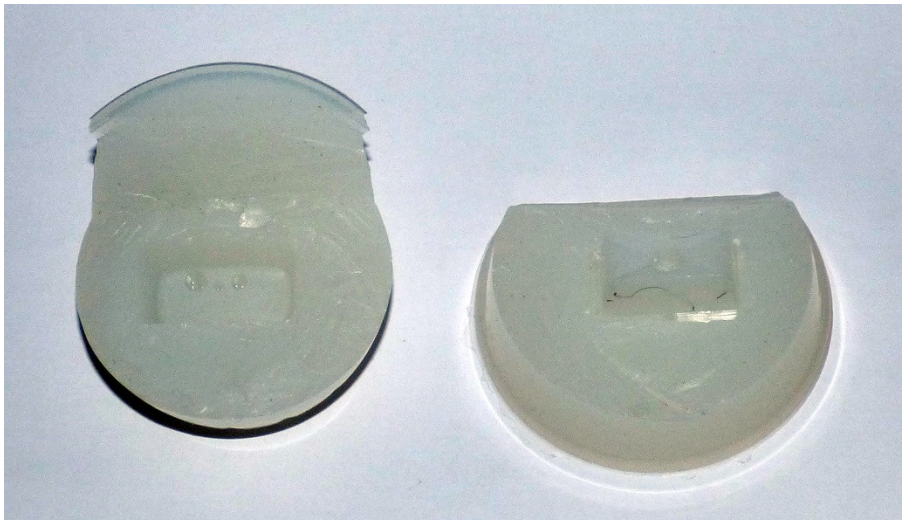

Figure 9: The silicon rubber mold for the key cap was cast in one piece and later cut open with a sharp knife.

Figure 10: The key cap as cast in clear epoxy resin with the casting spruce removed and slightly wet sanded.

Next I cast the new cap using clear Epoxy resin. After filling some small bubbles and sanding several layers of a matching Humbrol plastic model aircraft paint were applied, wet sanding the surface between these coats.

For the key label, I bought a few sheets of laser printable water slide paper and printed the label in slightly varying sizes with my laser printer. I used a very thin slide paper and carefully applied the decal. After letting the decal dry for 24 hours, I spray-painted the surface with several layers of clear lacquer to avoid rubbing the label off. Unfortunately, I was impatient and did not wait long enough between the layers, so that the lower clear layer started to crinkle and I had to wet sand the cap before adding another coat. However, in the end, after several days of surface finishing, the result was very nice – a satin gloss finish, similar to the original key caps and just the right color.

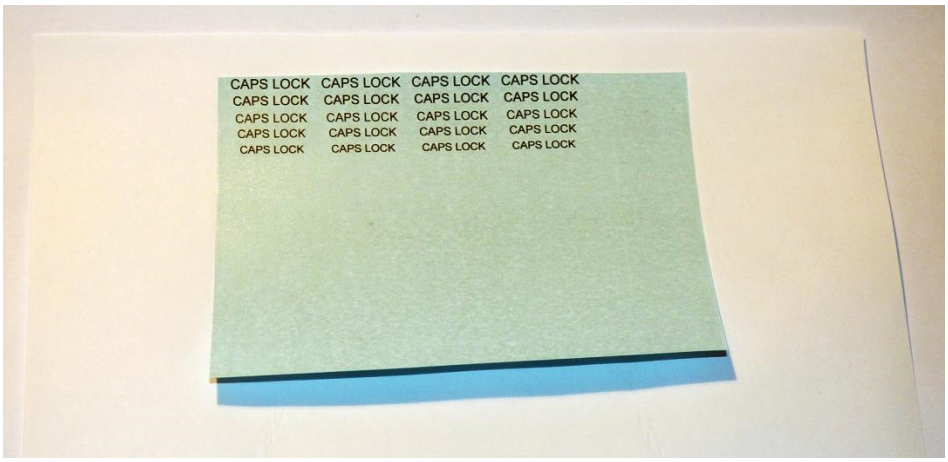

Figure 11: To minimize waste, I fixed a piece of decal paper to a sheet of support paper with two squares of thin double sided tape.

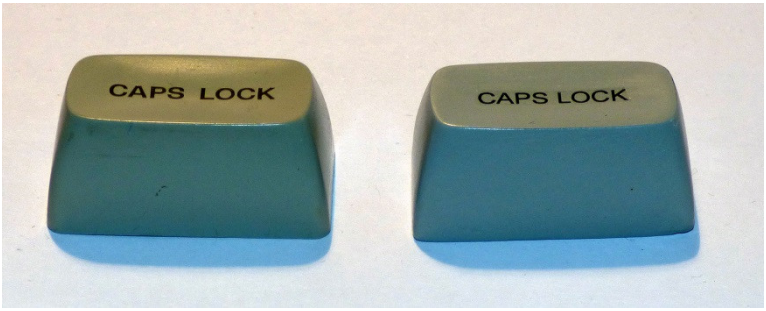

Figure 12: Key caps: left original, right: reproduction, painted and with decal applied, but not yet coated with clear protective layer.

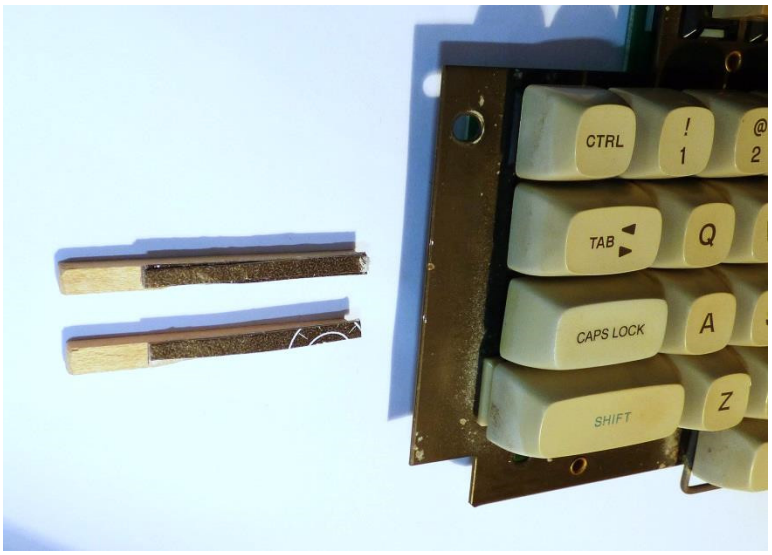

Figure 13: Two wooden square bars were adjusted with cardboard strips to support the key cap at the proper height exactly parallel to the base plate.

For mounting the key cap, I supported it by two wooden pegs of the right height, so that it would rest parallel to the black steel board. Additionally, thin cardboard strips were inserted above and below the cap to align it with its neighbors. I applied only a small amount of Epoxy resin to the steel pins and to the holes in the cap and after placing the key cap I inverted the arrangement to avoid any excess Epoxy flowing downwards towards the key switch.

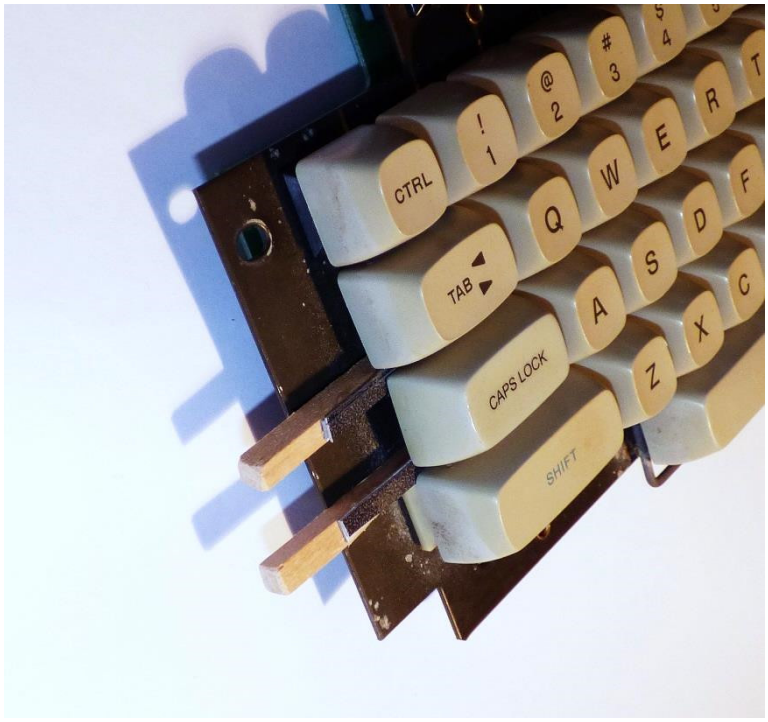

Figure 14: The key cap rests on the supporting bars while the epoxy cures.

Figure 15: The final key cap fits nicely into the keyboard, is difficult to detect and works fine.

Figure 16: The keyboard mounted in the HP 9836 in its natural habitat.

The PSU

Next, while I was toying around to determine the memory configuration and the mass storage msus syntax, the machine suddenly died. You know that sinking feeling when this happens. What did I do? Should I have kept the machine as a pure exhibition piece? No – I want to be able to use and explore my systems.

The 16A low voltage fuse had blown. After replacing the fuse it instantly blew again.

So I pulled out Tony Duell's wonderful schematics (with all its glorious 186 pages!) and the Service Manual. Following the Manual, the solution was simple: "replace the regulator board" – not really an option for me.

Compared to other HP designs, the power supply is relatively simple. It produces only +5, +12 and -12 Volts. A massive boat anchor of a transformer powers a rectifier board which feeds about 30 Volts into a large buffer capacitor. From there, a regulator board contains three regulators for the voltages and a crowbar over-voltage protection circuit.

I feared that a silicon component in one of the three voltage regulator circuits had burnt and hoped that no over-voltage had propagated to the core of the machine (assuming that crowbar and fuse had done their work).

Studying the schematics and the service manual helped to identify the correct edge connector pins on the regulator board. I found that the power input of the regulator board was completely shorted. A visual inspection showed no signs of heat or leaking capacitors.

First, I suspected a permanent short in the thyristor in the crowbar circuit. Desoldering and testing proved that it was good. Next in the input were capacitors C20 (680 µF electrolytic) and C27 (100 nF ceramic) both between input voltage and ground. I remembered that I had noticed a very faint fishy smell when I sniffed across the board the first time, but now I was not sure. Anyway, after desoldering capacitor C20 the short was gone. And the underside of the capacitor did not look nice – obviously it had leaked a long time ago and the electrolyte had accumulated and dried up on its underside. I also replaced the second capacitor C10 of the same size and make. The remaining capacitors looked fine and tested good, so I did not replace them.

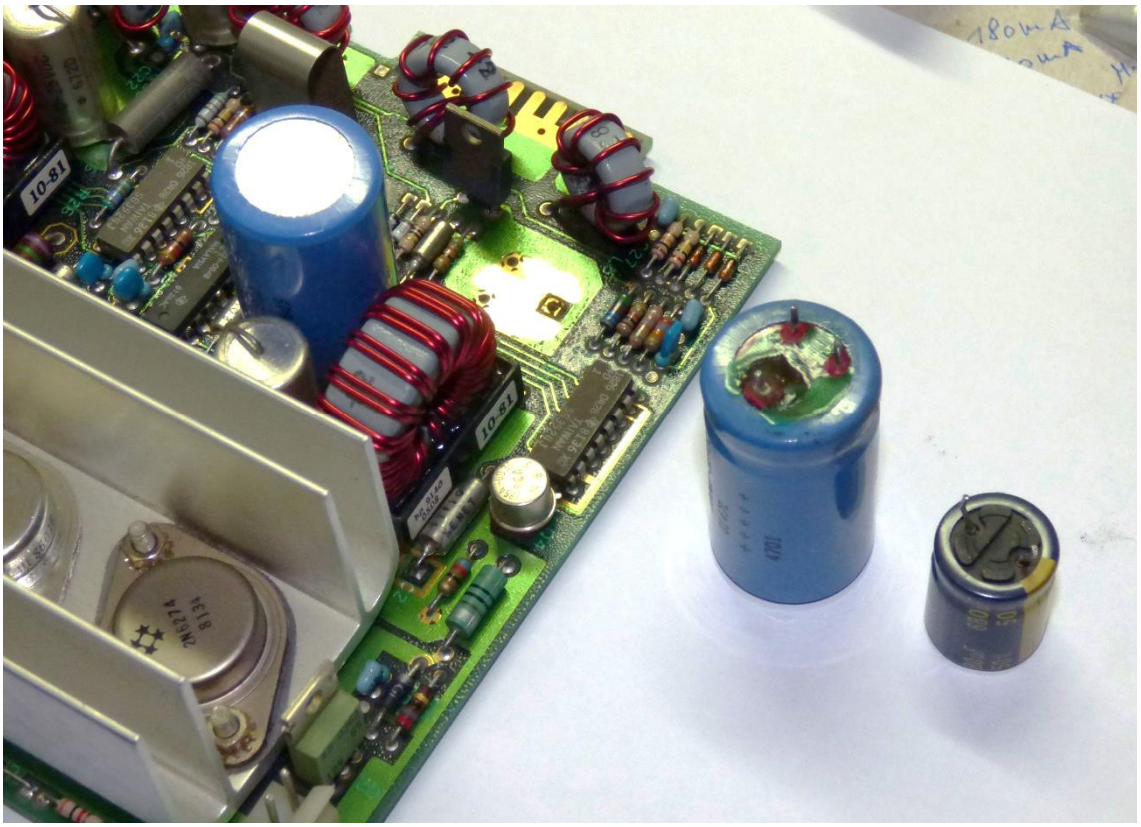

Figure 17: PSU regulator board with defective capacitor removed and modern replacement. The other blue capacitor was also replaced.

Luckily, there was no visible corrosion on the PCB. I replaced the capacitor with a new one which I had in my drawers. The modern type was much smaller and had a smaller pin distance so the wires had to be bent slightly to fit the hole pattern on the PCB. Also mine had only two legs (I don't even know, whether three pronged devices are manufactured anymore).

Anyway, after cleaning the board with isopropanol, to make sure no corrosive substances were left, I soldered the new capacitor in and the short was, of course, gone. Testing the regulator board showed the proper output voltages and after reinserting it into the mainframe the system booted up again. Joy!

The second Sprague electrolytic capacitor of the same type was replaced later, even if it still tested good.

So, in this case, as has already been demonstrated by many other repairs, the old electrolytic capacitors were the problem again.

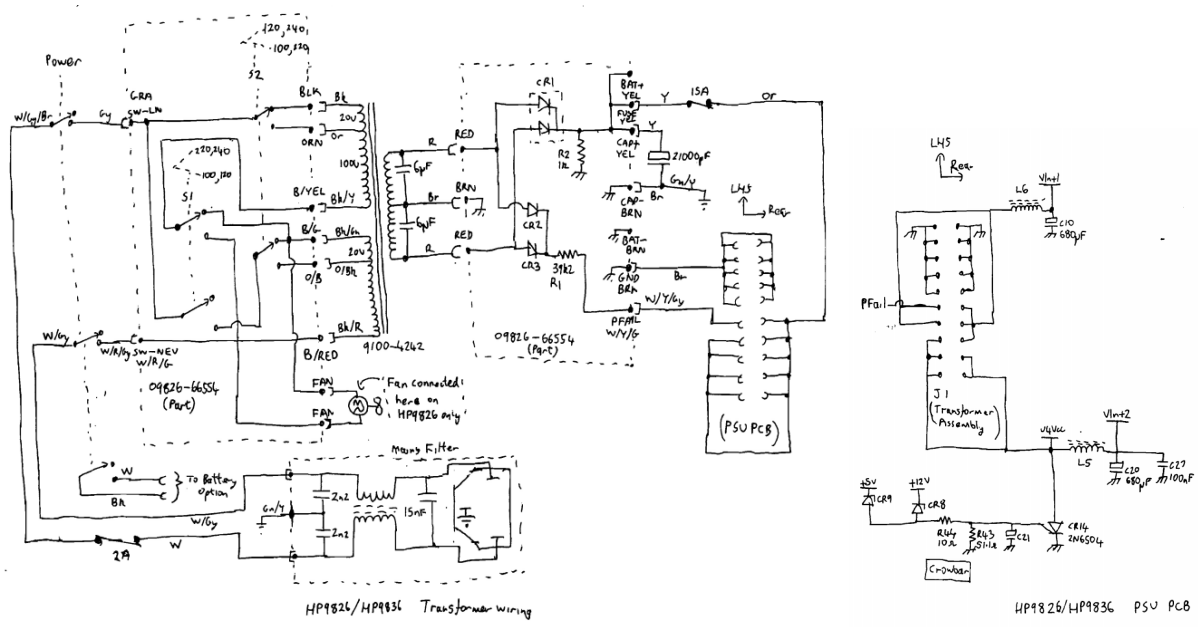

Figure 18: Tony Duell's schematics of the PSU with transformer, rectifier board and fuses. The right hand side shows the input section of the regulator board with its crowbar circuit. The culprit was C20 in the lower right of this figure. Note that C10 in the upper right is of the same type and was replaced too.

And here comes the HP 9836CU

A few months after I obtained my 9836A, I stumbled across a HP 9836C on E-Bay which I found very interesting, but it went for a ridiculous price of more than 400€.

Just a few weeks later, another HP 9836, in this case even a “CU” model complete with its color monitor was offered by a commercial scrapper. It did not look too promising because the HP-IB cables and even the short monitor cable had been cut for the copper. Only the connectors were still attached to the system. Obviously only a few people wanted to have this machine and I obtained it for 185€ including shipping (which caused the seller some headache, as the whole package weighs over 40 kg).

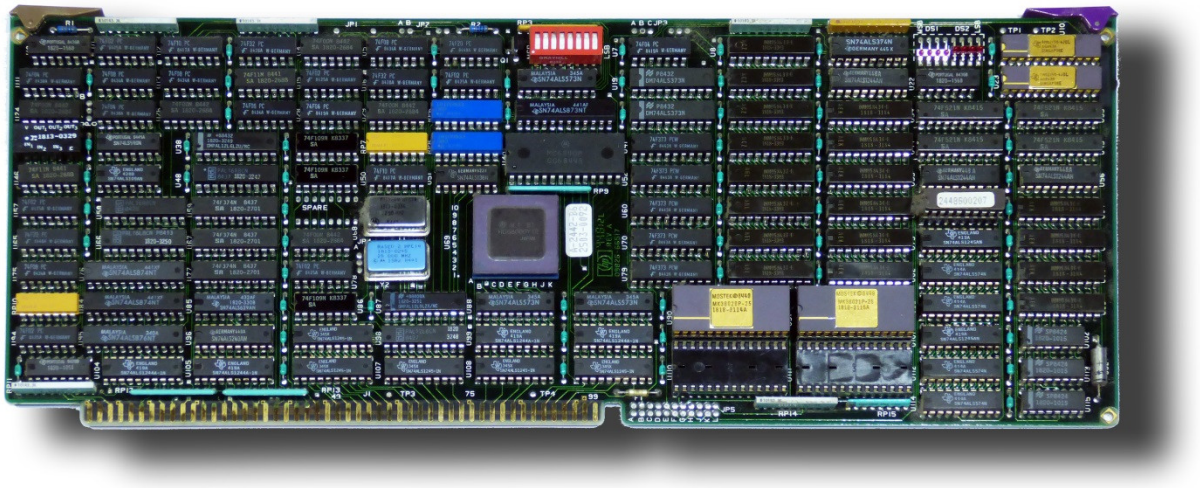

Figure 19: The 09826-66517 CPU board of the 9836CU with the MC68000R12 CPU, boot ROMS 3.0, a few PALs and 18 4Kx4 SRAM chips. The uneven looks of the gold fingers are due to poor lighting, they are in good condition.

The system included an Eventide WKBP-16 RAM board with 1 MB, a 98628A Opt.100 Datacomm and a 98622 GPIO board and as a bonus the math coprocessor 98635A board.

Repairing the Key Switches

I found that 3 key switches were completely broken off and one was just hanging dearly on to the steel key board plate. At least all parts including the key caps were present. In this case, the cherry switches were not broken at the stem, but the upper part of the switch case was ripped from the lower part.

Each mechanical switch consists of a plunger with a triangular wedge which operates a spring contact. The plunger is pushed up by a rather small helical spring of about 2 mm diameter. I glued the upper cases of the broken switches with a thin thread of steel filled epoxy to the lower cases. One has to be careful to avoid bringing glue into the switch mechanisms, but with a little bit of care and a toothpick this can be done. In one switch I had to replace the small spring which was crushed beyond repair. Luckily, I had a matching one in my “may be useful one day” box.

This time, the rotary encoder was still working and needed no attention.

Figure 20: Keyboard with broken switches taken off. Note that one of the function keys in the upper right is also almost broken and leans forward.

Figure 21: Enlarged view of a broken switch. The upper part of the switch case seems to be welded ultrasonically to the lower part and this connection breaks.

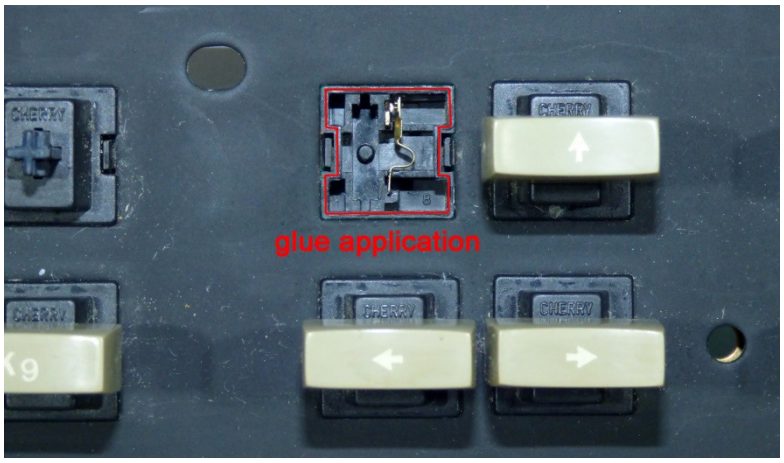

Figure 22: A thin thread of epoxy can be applied to the outer rim of the case and then the upper part including the plunger can be inserted carefully. Make sure that the small helical spring is in place (not yet installed on its pin in this picture) and that the pin on the plunger engages properly into the spring.

Making a new Video Cable

The scrapper had cut away the video cable and only one connector was still screwed to the monitor. This was unrepairable, so I had to build a new cable. The wiring is straight through, but the RGB signals should be individually shielded for good signal quality. I cut an old VGA cable and soldered its ends to male DB-15 connectors. I designed a hood for the rather thick cable and printed four identical semi-shells on my 3D-printer. I did not bother to add screws for closing the hoods; they are simply glued together with epoxy which also includes a cable restraint. The monitor side of the cable received the two original screws with washers to fasten the connector to the monitor. At the other end I inserted two countersink head screws from the connector side into the hood and secured them to the hood with a blob of epoxy. This allows pulling the cable hood together with the DSUB connector from the female connector.

Figure 23: The new video cable and the sad remains of the original cable..

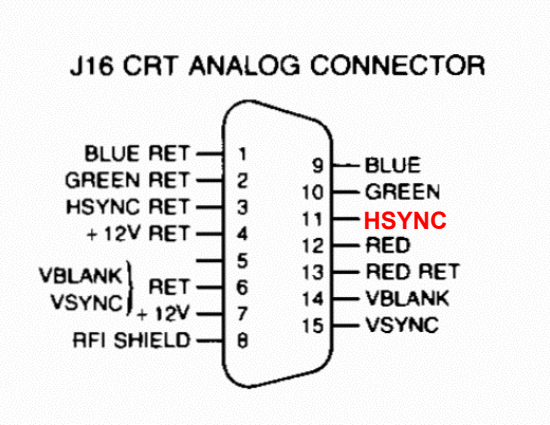

Figure 24: Simple straight through wiring of the color video cable lifted from the service manual. Note that pin 11 carries HSYNC which is not labeled in the HP document. The matching return wire 3 is labeled, though. I connected the ground pins 3 and 6 to a common ground wire as my VGA cable did not have more wires. Only pin 5 is not connected. The 12 V signal is used to switch the monitor on and off.

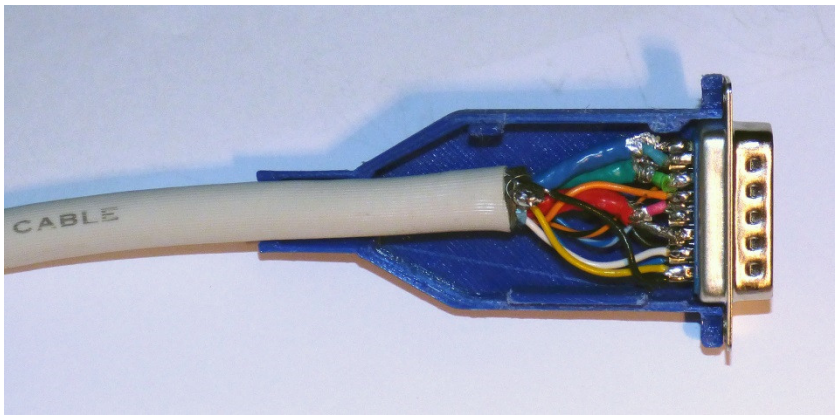

Figure 25: A look under the hood of the new cable before gluing it together. The RGB wires are shielded and not very convenient to solder to the connector.

Figure 26: The new cable installed. The upper connector is secured to the monitor with two screws. The lower connector (which should have a sliding lock), is just held in place by friction.

And the Rest

After cleaning lots of fluff from the inside of the machine, cleaning and lubricating the floppy disc drives I tried to boot the machine after removing all boards from the DIO cage. The monitor was not attached to the system. The self-test stopped immediately after the first LED sweep sequence with a

0100 0100 pattern. This indicated that not even the minimum 16 KB of RAM could be found. I thought that the CPU board should carry 128 KB or RAM. But thanks to Paul Berger I learned that the SRAM chips on the CPU board are merely cache and buffer RAMs for CPU and MMU. So I added the 1 MB Eventide RAM board and indeed the boot sequence passed all tests. Without a monitor and without diskettes in the drives, the boot sequence stopped with one LED on the floppy controller lit. After I added a BASIC 2.0 ROM board the system seemed to boot and no LED stayed illuminated.

In the meantime I had the new video cable ready and added the monitor to the system. I was very much delighted to see the green text of the boot screen and finally the BASIC 2.0 prompt.

When I tried the color graphics commands I learned that the extensions AP 2.1 are necessary to use color. In case of later BASIC versions, the GRAPHX extension has to be loaded.

The math coprocessor board 98635A is an interesting device and more information can be found in the Pascal System Designer's Guide (98615-90074). Its NS 32081 FPU offers the four basic operations on IEEE floating point numbers. The chip does not provide trigonometric or logarithmic operations. Its eight registers are memory mapped and it should be possible to control it directly from BASIC with some assembler routines. Maybe it is time for another Mandelbrot routine?

First Steps with BASIC 2.0 on the HP 9836

This early version of HP BASIC is missing many features of the later BASICs but it still quite useable. Because it is on my ROM board it boots immediately, which is very nice for the early 9826 and 9836 machines.

Mass Storage

The 9836 system has two 5-¼" diskette drives and the ROM BASIC 2.0 can also talk to external AMIGO drives. The right hand drive is `":INTERNAL,4,0"` and the left hand drive `":INTERNAL,4,1"`.

The default drive can be set with a `MASS STORAGE IS` command, `MASS STORAGE IS ":INTERNAL"` defaults to drive 4,0, i.e. the right hand drive. The left hand drive can be select as default by issuing `MASS STORAGE IS ":INTERNAL,4,1"`.

`CAT ":INTERNAL"` lists the files on the default MSUS, `CAT ":INTERNAL,4,0"` the ones on the right and `CAT ":INTERNAL,4,1"` the files on the left drive.

Copying a file from the default to the left disk drive `COPY "FILE" TO ":INTERNAL,4,1"`.

Loading a file from the default drive `LOAD "FILE"` or from the left hand internal drive `LOAD "FILE:INTERNAL,4,1"`.

HP-IB Devices

The built-in HP-IB interface has the default select code 7. Thus a listing of the current BASIC program can be sent to an external printer with HP-IB address 1 with `LIST #701`.

When it comes to disk drives, you can access drives supporting the Amigo protocol with the identifier `HP8290X` (for 9121S, 9121D, 9133 floppy), `HP9895` (for 9133 hard disk, 9895M and 9896S) or, `HP82901` (for 82901M and 82901S) or `HP82902` (for 82902M). Here 9133 means the early 9133 disk drive, not the later one with CS80 protocol often used with HP 9000 systems.

HPDRIVE can, for example, simulate the 9895 AMIGO diskette drive.

`LOAD "FILE:HP9895,7,1"`

Note: on my PC system, HPDRIVE must be run without the –d flag otherwise it is too slow to complete e.g. the INITIALIZE command in time.

For accessing more advanced disks than the classical AMIGO drives one has to load the AP2.1 extensions.

`LOAD BIN "AP2_1"`

These extensions add the `CS80`, `HP9133`, `HP9134` and, `HP9135` protocol specifiers to the MSUS string. Here, 9133 stands for the later disk drive model.

An external CS80 disk drive having HPIB Address 3 and unit number 0 can then be accessed as

`MASS STORAGE IS ":CS80,703,0"`

`CAT ":CS80,703,0"`

`LOAD "FILE:CS80,703,0"`

BASIC 2.0 Programs

The command `EDIT` enters edit mode where the cursor and line manipulation keys as well as the knob can be used. This command is also on one of the function keys in the upper right of the keyboard.

Listing a file on the printer having HPIB address 1

`LIST #701` or with a range of lines `LIST #701,100,200`.

The knob can be used to move quickly in the editor, the `SHIFT` key switches x vs. y direction.

PHYREC Binary Program

This CSUB contains two keywords to read or write a sector of 256 bytes (0…127 16-bit integers).

```
DIM Sector(127)
INTEGER  Nsector
Nsector=0
Phyread Nsector, Sector(*)
print Sector(0) DIV 255;Binand(Sector(0),255)

Phywrite Nsector, Sector(*)
```

Using READIO and WRITEIO

Arbitrary memory locations can be accessed byte-wise by using the special identifier 9826

```
Address = &H20000
Bdata = READIO ( 9826, Address )
WRITEIO 9826, Address; Bdata
```

For accessing memory 16-bit word-wise the same special identifier is used, but with a negative sign

```
Address = &H20000
Wdata = READIO ( -9826, Address )
WRITEIO -9826, Address; Wdata
```

The address of numeric variables can be found by reading with the special identifier 9827

```
Integer Codedata(32)
Caddress = READIO ( 9827, Codedata(1))
```

Unfortunately it is not possible to obtain the address of a string variable with this function. However, by embedding the string variable into a common block it is possible to access its contents.

Note that the variables in common blocks are stored in reverse order, from low to high addresses. Therefore, in the following dump example, we have to start at the address of the last item of the COM block.

The common block

```
10    COM /Common/ INTEGER I1,I2,L$[8],INTEGER I3,I4,REAL R1,INTEGER Last
```

is actually stored as

```
start length item
  0    2     Last
  2    8     R1
 10    2     I4
 12    2     I3
 14    2+8   L$[8]      - 2 bytes length, 8 bytes characters
 24    2     I2
 26    2     I1
```

Common block dump example:

```
10    COM /Common/ INTEGER I1,I2,L$[8],INTEGER I3,I4,REAL R1,INTEGER Last
20    I1=1
30    I2=2
40    I3=3
50    I4=4
60    R1=1.0E-12
70    L$="ABCD"
80    Last=32767
90    !
100   Addr=READIO(9827,I1)
110   PRINT "I1 at ";Addr
120   Addr=READIO(9827,I2)
130   PRINT "I2 at ";Addr
140   Addr=READIO(9827,I3)
150   PRINT "I3 at ";Addr
160   Addr=READIO(9827,I4)
170   PRINT "I4 at ";Addr
180   Addr=READIO(9827,Last)
190   FOR I=1 TO 14
200    B=READIO(-9826,Addr)
210    B1=READIO(9826,Addr)
220    B2=READIO(9826,Addr+1)
230    PRINT USING "DDDDDDDD,X,A,DDDDDD,X,A,X,DDD,X,DDD";Addr,":",B,"=",B1,B2
240    Addr=Addr+2
250   NEXT I
260   END

RUM

I1 at -19394
I2 at -19396
I3 at -19408
I4 at -19410
  -19420 : 32767 = 127 255   - Last: 1 word,  2 bytes
  -19418 : 15729 =  61 113   - R1:  4 words, 8 bytes
  -19416 :-26727 = 151 153
  -19414 :-32467 = 129  45
  -19412 : -5615 = 234  17
  -19410 :     4 =   0   4   - I4 = 4
  -19408 :     3 =   0   3   - I3 = 3
  -19406 :     4 =   0   4   - 4 characters used in L$[8]
  -19404 : 16706 =  65  66     8 bytes with content of L$ 'A','B'
  -19402 : 17220 =  67  68                               'C','D'
  -19400 :     0 =   0   0       empty part of string
  -19398 :     0 =   0   0
  -19396 :     2 =   0   2   - I2 = 2
  -19394 :     1 =   0   1   - I1 = 1
```

Writing to the identifier 9827 performs a jump to a subroutine (jsr) at the given address.

```
WRITEIO 9827, Caddress; DOdata
```

Here, `Caddress` would be the address of an array with words of machine code, ending in a return from subroutine (rts) instruction. The additional parameter `D0data` is placed in the processor register D0 so that e.g. the address of a buffer can be transferred.

The following example shows a minimal machine language routine which increments the 16-bit word (a BASIC INTEGER) at the memory address given in D0data.

```
Integer CodeBuffer(10)
Integer Databuffer(1)
!
! 48E7 FFFF  MOVEM.L  D0-D7/A0-A6,-(SP)         ; save registers
! 2040       MOVE.L   D0,A0                     ; to address register
! 5250       ADDQ.W   #1,(A0)                   ; increment 16-bit value by 1
! 4CDF FFFF  MOVEM.L  (SP)+,D0-D7/A0-A6         ; restore registers
! 4E75       RTS                                ; return
!
DATA 48E7,FFFF,2040,5250,4CDF,FFFF,4E75,STOP
!
RESTORE
I=0
Nextword: READ Word$
 IF Word$="STOP" THEN GOTO Done
 Codebuffer(I) = IVAL(Word$,16)
 I=I+1
GOTO Nextword
Done: MaxWords=I-1
!
Caddress = READIO ( 9827, Codebuffer(0))
Daddress = READIO ( 9827, Databuffer(0))
!
Databuffer(0) = 0
PRINT Databuffer(0)
FOR I=1 TO 10
 WRITEIO 9827, Caddress; Daddress
 PRINT Databuffer(0)
NEXT I
END
```

The Alpha screen data starts at 0x512000 and is 4 Kbytes long. It is organized in 16-bit words per character The odd numbered addresses contain the actual character code and the even addresses the character attributes (bit 3=half bright).

The graphics screen RAM of the "A" model starts at 0x530000.

The early BASIC versions do not have functions for accessing graphics RAM e.g. for bitmap operations. Only `GSTORE` and `GLOAD` for storing resp. loading the entire screen are available.

Using `READIO` and `WRITEIO`, it is possible to access any byte in the graphics RAM.

The code fragment below writes some patterns directly to the graphics RAM.

```
! HP 9836, monochrome
! 512 pixels = 64 bytes per row
! 390 rows per screen
INTEGER X, B
! first, left byte of upper row
Address = 5439488
! draw a dotted horizontal line, 170d = 10101010b
B = 170
FOR X=0 TO 63
 WRITEIO 9826, Address+X; B
NEXT X
! skip to start of bottom row
Address = Address + (390-1)*64
! draw a dotted horizontal line with words
B = 170*256 + 170
FOR X=0 TO 31
 WRITEIO -9826, Address+X; B
NEXT X
```

```
END
```

If you use `GLOAD` and `GSTORE` with a multi-dimensional array to load or store the complete display RAM, remember that HP BASIC (like FORTRAN) increments the rightmost index first. So the dimension of an `INTEGER` array for 64 bytes in 390 lines of the monochrome 9836 display would be

```
INTEGER Screen(1:390,1:32)
```

	Model 216	Model 217	Model 226	Model 236A	Model 236C	Model 237
Width (mm)	168	230	130	210	210	312
Height (mm)	126	175	100	160	160	234
Width (pixels)	400	512	400	512	512	1024
Height (pixels)	300	390	300	390	390	768
Pixels/mm	2.38	2.23	3.08	2.44	2.44	3.28
mm/pixel	0.42	0.45	0.33	0.41	0.41	0.30
Start address	$530001	$530000	$530001	$530000	$520000	$300000
Last pixel address	$537531	$536180	$537531	$536180	$550BFF	$3BFFFE
Ending address	$537FFF	$537FFF	$537FFF	$537FFF	$550BFF	$3FFFFE
Addressed Memory	$7FFF	$7FFF	$7FFF	$7FFF	$30C00	$FFFFF
Actual Memory	$3FFF	$7FFF	$3FFF	$7FFF	$18600	$20000
Visible memory	$3A98	$6180	$3A98	$6180	$30C00	$18000
Address layout	7	8	7	8	9	10

Table 1: Characteristics of the graphics RAM of various HP 9000/200 systems [1]. Address layout 7 uses only the odd bytes, layout 9 corresponds to 4 bit indices into the color map and layout 10 is one byte per pixel (bit 0 used).

The following example code demonstrates two versions of a simple `Bplot` subroutine for the HP 9836 with monochrome monitor, constructed from the information given above.

The first version is written in pure BASIC, whereas the second version makes use of a short machine language routine, embedded into a BASIC subroutine.

Version	Time
BASIC 2.0	2.110 s
Machine Language	0.120 s

Table 2: Run times of both Bplot versions.

For simplicity, the `X`-position will always be byte aligned. No precautions have been taken to avoid out-of-screen writes. Appropriate tests could be added to the `Bplot` routines.

```
10    !
20    ! Requires AP2.1
30    !
40    ! Martin Hepperle, 2022
50    !
60    INTEGER X,Y,Wb
70    DIM Buffer$[80]
75    !
80    ! load machine language routine into COM
90    CALL Bplot_init
95    !
100   ! get logo bitmap
110   Buffer$=FNLogo$
```

```
115   !
120   TO=TIMEDATE
130   GCLEAR
140   WINDOW 0,511,0,389
150   MOVE 466,0
160   DRAW 466,389
170   MOVE 510,0
180   DRAW 510,389
190   X=474
200   Wb=4
210   FOR Y=8 TO 360 STEP 32
220     CALL Bplot(X,Y,Wb,Buffer$)
230   NEXT Y
240   T1=TIMEDATE
250   PRINT "dT=";T1-TO
260   END
270   ! --------------------------------
280   ! Load the ML program
290   SUB Bplot_init
300     COM /Bplot/ INTEGER Code(0:39),Bitmap$[100],INTEGER Xb,Yb,Wbytes
310     INTEGER I
320     DIM Word$[4]
330     DATA 48E7,FFFF,2040,3218,3418
340     DATA ED42,3618,E64B,3818,88C1
350     DATA 2A3C,0053,0000,DA43,DA42
360     DATA 2245,4283,B644,6700,001C
370     DATA 4285,B245,6700,000A,1398
380     DATA 5000,5245,60F2,D3FC,0000
390     DATA 0040,5243,60E0,4CDF,FFFF
400     DATA 4E75
410     DATA STOP
420   !
430     RESTORE
440     I=0
450 Nextword:READ Word$
460     IF Word$="STOP" THEN SUBEXIT
470     Code(I)=IVAL(Word$,16)
480     I=I+1
490     GOTO Nextword
500   !
510   SUBEND
520   ! --------------------------------
530   ! Bit Plot
540   SUB Bplot(INTEGER X,Y,Bytes_per_row,Buffer$)
550     COM /Bplot/ INTEGER Code(0:39),Bitmap$[100],INTEGER Xb,Yb,Wbytes
560   ! Copy to COM
570     Xb=X
580     Yb=Y
590     Wbytes=Bytes_per_row
600     Bitmap$=Buffer$
610   ! get addresses
620     Dataaddr=READIO(9827,Wbytes)
630     Codeaddr=READIO(9827,Code(0))
640   ! call ML routine
650     WRITEIO 9827,Codeaddr;Dataaddr
660   SUBEND
670   ! --------------------------------
680   DEF FNLogo$
690   INTEGER X,Y,Wbytes
700   DIM Bitmap$[80]
710   ! Definition of bitmap data
720   ! 4 bytes per line, 16 lines
730   DATA 4,18
740   ! top to bottom
750   DATA 63,255,255,252,127,255,255,254
760   DATA 255,240,15,255,255,240,3,255
770   DATA 255,176,1,255,255,62,124,255
780   DATA 255,63,126,255,254,51,102,127
790   DATA 254,51,102,127,254,51,102,127
800   DATA 254,51,102,127,255,51,126,255
810   DATA 255,51,124,255,255,128,97,255
820   DATA 255,192,99,255,255,240,111,255
830   DATA 127,255,255,254,63,255,255,252
840   !
850   ! Read bitmap to transfer buffer
```

```
860   READ Wbytes
870   READ Nrows
880   Bitmap$=""
890   FOR I=1 TO Nrows*Wbytes
900   READ C
910   Bitmap$=Bitmap$&CHR$(C)
920   NEXT I
930   RETURN Bitmap$
940   FNEND
```

Listing 1: The same program adapted for using a machine language subroutine.

```
48E7 FFFF               movem.l d0-d7/a0-a7,-(sp)

                ; d0:  address of Last in COM
                ;      addq.w #2,d0
                ; a0:  address of WB in COM
2040                    move.l d0,a0

                ; d1:  WB in COM
3218                    move.w (a0)+,d1
                ; d2:  Y in COM
3418                    move.w (a0)+,d2
                ; d2:  Y*64 in COM
ED42                    asl.w  #6,d2

                ; d3:  X in COM
3618                    move.w (a0)+,d3
                ; d3:  X/8 in COM
E64B                    lsr #3,d3

                ; d4:  string length
                ; a0:  start of string
3818                    move.w (a0)+,d4
                ; d4:  d4/d1 = Rows
88C1                    divu.w d1,d4

                ; d5:  destination address, upper left
2A3C 00530000           move.l #5439488,d5
DA43                    add.w  d3,d5
DA42                    add.w  d2,d5
                ; a1:  destination
2245                    move.l d5,a1

                ; d3:  row=0
4283                    clr.l d3

                ; WHILE Row  while d3<d4
                WhileRow:
B644                    cmp.w   d4,d3
6700 001C               beq     EndWhileRow

                ;   Byte=0
4285                    clr.l   d5
                ;   WHILE Byte  while d5<d1
                WhileByte:
B245                    cmp.w   d5,d1
6700 000A               beq     EndWhileByte

                ;   copy source byte to destination
1398 5000               move.b  (a0)+,(a1,d5)

                ; END WHILE Byte
5245                    addq.w  #1,d5
60F2                    bra WhileByte

                EndWhileByte:
D3FC 00000040           add.l  #64,a1

                ; END WHILE Row
5243                    addq.w  #1,d3
60E0                    bra WhileRow

                EndWhileRow:
4CDF FFFF               movem.l (sp)+,d0-d7/a0-a7
```

```
4E75                rts
```

Listing 2: This Bplot code has been embedded into the BASIC routine Bplot_init above.

What about Speed?

Of course, I had to run the infamous BYTE benchmark "Eratosthenes Sieve" on my HP 9836. Three variants of the same algorithm were implemented and the results are listed below.

The assembler version was my first 68000 program ever and is therefore not perfect, but produces the correct results. It shows how one can use small assembler routines inside BASIC programs without resorting to CSUBs or third party assembler tools. I developed the code on my PC using the Easy68K assembler and simulator for debugging and then typed the machine language words into the BASIC editor.

interpreted BASIC 2.1	180 s
compiled Pascal 3.25	9.9 s
68000 assembler, in BASIC wrapper	2.4 s

Table 3: Eratosthenes Sieve benchmark. Execution times are for 10 iterations,

For comparison: BYTE Magazine gives a time of 5.9 s for a HP 9830 (HP Pascal 1.0 on its 68000 @ 8 MHz). A HP 85 with its Capricorn @ 640 kHz and interpreted BASIC takes 3084 s – its machine language version runs in 21 s. An IBM PC with interpreted BASICA needs about 1900 s.

```
10    INTEGER Flags(8191)
20    INTEGER M,I,K,Prime,Count
30    T0=TIMEDATE
40    FOR M=1 TO 10
50     PRINT M
60     Count=0
70     FOR I=0 TO 8190
80      Flags(I)=1
90     NEXT I
100    FOR I=1 TO 8190
110     IF Flags(I)=0 THEN GOTO 190
120     Prime=I+I+3
130     K=I+Prime
140     WHILE K<=8190
150      Flags(K)=0
160      K=K+Prime
170     END WHILE
180     Count=Count+1
190    NEXT I
200   NEXT M
210   PRINT Primes;" Primes in ";TIMEDATE-T0;" seconds"
220   END
```

Listing 3: The Sieve program in pure BASIC performs 10 iterations.

```
0000                    *-------------------------------------------------------------
0000                    * BYTE Eratosthenes Sieve Benchmark
0000                    * Martin Hepperle, 6/2022
0000                    * 68000 assembler code
0000                    * Call with address of a 8191 bytes array in register D0
0000                    * On return array[0] will have the count value of 1899
0000                    *-------------------------------------------------------------
0000  =00001FFE         SIZE   equ   8190
0000                    ;
0000                    entry:
0000                    ; save all to be sure - probably already done by HP BASIC
0000  48E7 FFFF                 movem.l d0-d7/a0-a7,-(sp)
0004
```

```
0004                    ; on entry:
0004                    ; D0:  address of flags[SIZE] byte array
0004
0004                    ; Register Usage:
0004                    ; D0:   address of flags byte array
0004                    ; D1:   i loop counter
0004                    ; D2    count
0004                    ; D3    prime
0004                    ; D4    k
0004                    ; A0    address of flags[i]
0004                    ; D5,A1 address of flags[k]
0004                    ;
0004                    ; initialize flags[0..SIZE] with true
0004  2040                      move.l D0,A0
0006  323C 1FFD                  move.w #SIZE-1,D1
000A  10FC 0001           Fill:   move.b #1,(A0)+
000E  51C9 FFFA                  dbra   D1,Fill
0012
0012                    ; ---   count = 0
0012  4242                      clr.w D2
0014
0014                    ; D0: start address of flags byte array
0014  2040                      move.l D0,A0
0016
0016                    ;       i=0
0016  4241                      clr.w D1
0018                    ; main loop over flags[i]
0018                    NextNumber:
0018                    ; ---   if flags[i] == 1
0018  0C18 0001                 cmpi.b  #1,(A0)+
001C  6600 0024                 bne     Incr
0020
0020                    ;       begin
0020                    ; ---   prime = 3 + i  + i
0020                    ;       D3    = 3 + D1 + D1
0020  363C 0003                 move.w  #3,D3
0024  D641                      add.w   D1,D3
0026  D641                      add.w   D1,D3
0028                    ; ---   k  = prime + i
0028                    ;       D4 = D3    + D1
0028  3803                      move.w  D3,D4
002A  D841                      add.w   D1,D4
002C
002C                    ;       if k>SIZE goto Crossed
002C                    Crossing:
002C  0C44 1FFE                 cmpi.w  #SIZE,D4
0030  6E00 000E                 bgt     Crossed
0034
0034                    ; ---   flags[k] = 0
0034                    ;       (D0+D4)
0034  2A00                      move.l D0,D5
0036                    ;       add lower word
0036  DA44                      add.w  D4,D5
0038                    ;      to address register
0038  2245                      move.l D5,A1
003A  4211                      clr.b  (A1)
003C
003C                    ; ---   k  = k  + prime
003C                    ;       D4 = D4 + D3
003C  D843                      add.w   D3,D4
003E  60EC                      bra     Crossing
0040                    Crossed:
0040                    ;       count = count+1
0040  5242                      addq.w  #1,D2
0042
0042                    ;       end
0042                    Incr:
0042                    ;       increment loop counter i
0042  5241                      addq.w  #1,D1
0044                    ;       if I <= SIZE then goto Next
0044  0C41 1FFE                 cmpi.w  #SIZE,D1
0048  63CE                      bls     NextNumber
004A
004A                    ; place count into integer at flags(0) so that BASIC can see
004A  2040                      move.l D0,A0
004C  3082                      move.w D2,(A0)
```

```
004E
004E                    ; restore all - probably also done by HP BASIC
004E  4CDF FFFF             movem.l (sp)+,d0-d7/a0-a7
0052  4E75                  rts
0054                    ;
0054                        END   main
```

Listing 4: The assembled single iteration Sieve code with the resulting machine code.

```
10    !
20    ! Requires AP2.1
30    !
40    ! Martin Hepperle, 2022
50    !
60    INTEGER Codebuffer(128)
70    INTEGER Databuffer(8190)
80    REAL Caddress
90    REAL Daddress
100   ! Eratosthenes Sieve Machine Code Words
110   DATA 48E7,FFFF,2040,323C,1FFD,10FC,0001,51C9,FFFA
120   DATA 4242,2040,4241,0C18,0001,6600,0024,363C,0003
130   DATA D641,D641,3803,D841,0C44,1FFE,6E00,000E,2A00
140   DATA DA44,2245,4211,D843,60EC,5242,5241,0C41,1FFE
150   DATA 63CE,2040,3082,4CDF,FFFF,4E75,0000
160 !
170 !
180   RESTORE
190   I=0
200 Nextword:  READ Word$
210            IF Word$="0000" THEN GOTO Done
220            Codebuffer(I)=IVAL(Word$,16)
230   I=I+1
235 ! TODO: should test for Codebuffer() overrun
240   GOTO Nextword
250 Done:   Maxwords=I-1
260   !
270   Databuffer(0)=0
280   ! Get Addresses
290   Caddress=READIO(9827,Codebuffer(0))
300   Daddress=READIO(9827,Databuffer(0))
310   PRINT "Code:";DVAL$(Caddress,16)
320   PRINT "Data:";DVAL$(Daddress,16)
330   FOR I=0 TO Maxwords
340   PRINT USING 370;I,IVAL$(Codebuffer(I),16)
350   NEXT I
360   PRINT
370   IMAGE #,2D,":",4A,X
375 ! --- start of timing
380   T0=TIMEDATE
390   PRINT Databuffer(0)
400   FOR I=1 TO 10
410    WRITEIO 9827,Caddress;Daddress
420   NEXT I
430   PRINT Databuffer(0);"primes"
440   T1=TIMEDATE
445 ! --- end of timing
450   PRINT T1-T0
460   END
```

Listing 5: The BASIC program with machine code words performs 10 iterations too.

Connecting a “Centronics” Printer to the HP 9836

My HP 9836 did not have a parallel Centronics type interface, but I had a 98622A GPIO interface.

This interface is very common and has a wide 50-pin “Centronics” style female Amphenol plug. It supports 8- and 16-bit input and output via 16 dedicated I/O-lines. Additional control lines are available for handshaking. Switches allow selecting logic sense and handshaking options. Ideally you have a matching male connector with screw terminals and cable; otherwise you have to improvise with a 50-pin clip connector and additional screws. For these wide Amphenol connectors it is essential that the connectors are held firmly in place.

The other end of the cable was terminated by a female DB-25 connector, so that I can connect regular Centronics printer cables as used for IBM-PC systems. Alternatively, for directly plugging into a printer, you can of course attach a 36-pin male Amphenol connector to this end.

This simple cable works with my Epson MX and FX printers. Most of the actual work is to identify the correct wires inside the cable.

Note that the STROBE/ and ACK/ signals are not 100% Centronics compatible: they should be pulsed, but the timing of the falling edges obviously works with most printers.

Switch	0/1	Description
PCTL	1	invert, falling edge = STROBE/
PFLG	0	positive edge = ACK
PSTS	0	don’t care
HSHK	0	pulse mode
DIN	0	don’t care
DOUT	0	positive logic

Table 4: Settings on the GPIO interface.

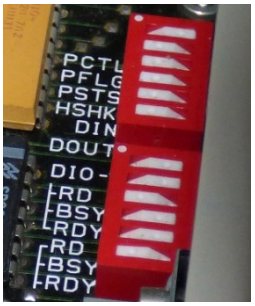

Figure 27: DIP switch settings.

My interface has a select code of 16 so that any `CAT` or `LIST` output can be printed easily by issuing a

```
PRINTER IS 16
```

Amphenol 50-pin	GPIO Signal	D-SUB DB-25	Amphenol 36-pin	Direction from I/F	Centronics Signal
17	DIO0	2	2	→	data bits
16	DIO1	3	3	→	
15	DIO2	4	4	→	
14	DIO3	5	5	→	
13	DIO4	6	6	→	
12	DIO5	7	7	→	
11	DIO6	8	8	→	
12	DIO7	9	9	→	
10	PCTL	1	1	→	STROBE/
44	PFLG	10	10	←	ACK
1	GND	18	33	–	

Table 5: Wiring the GPIO card to a Centronics cable.

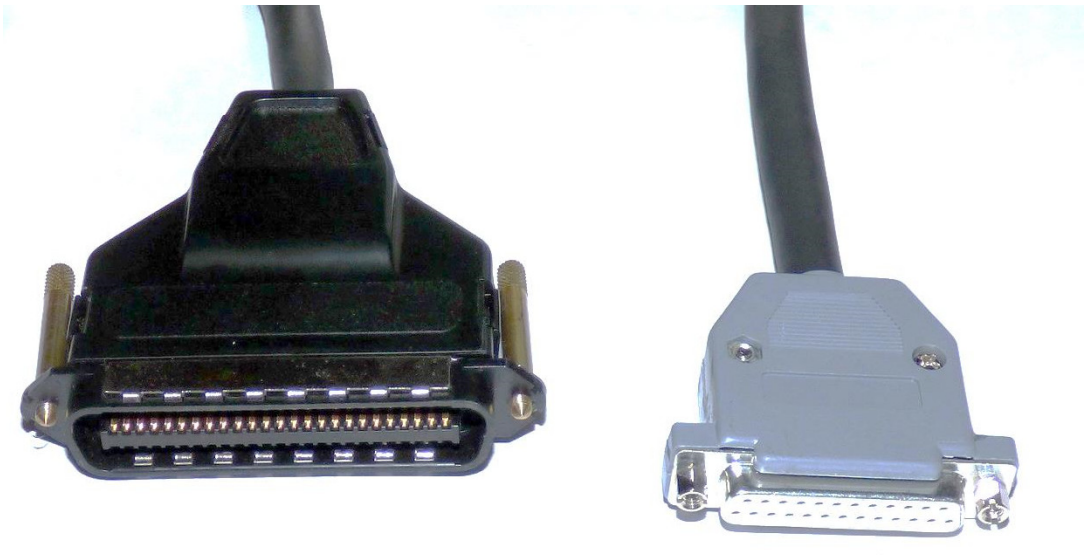

Figure 28: This Cable allows attaching a standard Centronics printer cable to the GPIO interface. The DB-25 connector has been equipped with hex nuts for securing the printer cable.

HP 9836 Screen Control

Control Codes

Chr$(7)	BEL	sound the keybords beeper
Chr$(8)	BS	backspace, not beyond first column of line
Chr$(10)	LF	move cursor down 1 line
Chr$(12)	FF	scroll screen up, print two blank lines, place cursor in first column of second line
Chr$(13)	CR	move cursor to first column of current line

Character Enhancement Codes

Bitmask

```
10001111
|   |||| bit 0    inverse
|   ||| bit 1     blinking
|   || bit 2      underline
|   | bit 3       half bright
bit 7             always 1
```

Chr$(128)	all enhancements off
Chr$(129)	inverse
Chr$(130)	blinking
Chr$(131)	invers and blinking
Chr$(132)	underline
Chr$(133)	underline and inverse
Chr$(134)	underline and blinking
Chr$(135)	underline, inverse, and blinking
Chr$(136)	half bright
Chr$(137)	half bright and inverse
Chr$(138)	half bright and blinking
Chr$(139)	half bright, inverse and blinking
Chr$(140)	half bright and underline
Chr$(141)	half bright, underline and inverse
Chr$(142)	half bright, underline and blinking
Chr$(143)	half bright, underline, inverse and blinking

Key Codes sent to Kbd after Chr$(255)

33	!	stop
73	I	clr I/O
35	#	clear line
37	%	clear from cursor to end of line
42	*	insert line at cursor
43	+	toggle insert character mode
45	-	delete character at cursor
47	/	delete line at cursor
60	<	←
62	>	→
71	G	shift → cursor to end of line
72	H	shift ← cursor to start of line
75	K	clear screen
76	L	toggle graphics
77	M	toggle alpha
86	V	↓ cursor down
84	T	shift ↓ cursor down
91	[	clear tab at cursor
93	]	set tab at cursor
94	^	↑ cursor up

87	W	shift ↑ cursor up
41	)	tab
40	(	shift tab
88	X	execute
69	E	enter
82	R	run
80	P	pause
67	C	continue

References

[1] HP 9000 Series 200 Computers “Pascal 3.0 System Designer’s Guide”, 98615-90074, February 1985 Edition 1.

[2] Duell, Tony, “9826-9836 Schematics”, 184 pages.